EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	628102	memory and (object or objects)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:36
L2	2059987	frame or frames or FID	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:37
L3	164252	1 and 2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:37
L4	3602460	object or objects	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:37
L5	1813092	release or released or releasing	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:38
L6	23117	4 near5 5	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:38
L7	8231	frame adj identifier or FID	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:38
L8	22	6 and 7	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:39
L9	24442795	@ad<"20030804"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:39
L10	20	8 and 9	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:39

EAST Search History

L11	5	3 and 10	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:42
L12	49609	(old or older or new or newer or aged or young or younger or age) with 2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:50
L13	49122	(assign or assigned or assigning or reassign or reassigned or reassigning) with (frame or frames or block or blocks)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:46
L14	86	(assign or assigned or assigning or reassign or reassigned or reassigning) with (stack) with (allocate or allocated or allocation or allocating)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:47
L15	0	12 and 13 and 14 and 7	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:47
L16	7	12 and 13 and 14 and (frame or frames)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:47
L17	6	9 and 16	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:47
L18	4	(US-6192517-$ or US-6047125-$ or US-5893121-$ or US-4422145-$).did.	USPAT	OR	ON	2006/06/22 18:49
L19	2886	((old or older or aged) with (frame or frames)) and (object or objects)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L20	0	14 and 19	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L21	2224	711/154.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51

EAST Search History

L22	1330	711/100.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L23	699	711/151.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L24	681	711/158.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L25	773	711/165.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:51
L26	2183	711/170.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:52
L27	6953	21 or 22 or 23 or 24 or 25 or 26	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:52
L28	7	14 and 27	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:52
L29	6	9 and 28	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:54
L30	4	(US-6505344-$ or US-6047125-$ or US-5491808-$ or US-5485613-$).did.	USPAT	OR	ON	2006/06/22 18:54
L31	7	18 or 30	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:55
L32	742	711/154.ccls.	US-PGPUB	OR	ON	2006/06/22 18:56
L33	546	9 and 32	US-PGPUB	OR	ON	2006/06/22 18:56
L34	1	19 and 33	US-PGPUB	OR	ON	2006/06/22 18:58

L35	0	14 and 33	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:58
L36	0	8 and 33	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2006/06/22 18:58

IEEE *Xplore* ®
RELEASE 2.1

Welcome United States Patent and Trademark Office

:▢:Search Session History

BROWSE　　　　**SEARCH**　　　　**IEEE XPLORE GUIDE**　　　　**SUPPORT**

Thu, 22 Jun 2006, 7:13:01 PM EST

Edit an existing query or compose a new query in the Search Query Display.

Search Query Display

Select a search number (#) to:

- Add a query to the Search Query Display
- Combine search queries using AND, OR, or NOT
- Delete a search
- Run a search

Recent Search Queries

		Results
#1	(((frame <and> object <and> (allocate <or> allocation)))<in>metadata)	30
#2	(((((frame <and> object <and> (allocate <or> allocation)))<in>metadata))<AND> (release <or> released <or> releasing<in>metadata))	0
#3	(((frame <and> object <and> (allocate <or> allocation)))<in>metadata)	30
#4	(((((frame <and> object <and> (allocate <or> allocation)))<in>metadata))<AND> (old <or> older <or> aged<in>metadata))	2
#5	(((frame <and> object <and> (allocate <or> allocation)))<in>metadata)	30
#6	(((((frame <and> object <and> (allocate <or> allocation)))<in>metadata))<AND> (reassign <or> reassigned <or> reassigning<in>metadata))	0

Help　　Contact Us　　Privacy & Security　　IEEE.org

Indexed by
🔳 Inspec®

THE ACM DIGITAL LIBRARY

🔖 Feedback Report a problem Satisfaction survey

Terms used **frame** and **object** and **allocate** or **allocation** and **old** or **older** or **aged**   Found 29,407 of 178,880

Sort results by [relevance ▽]   🔖 Save results to a Binder   Try an Advanced Search
                                                                                   Try this search in The ACM Guide
Display results [expanded form ▽]   ❓ Search Tips

                                           🗔 Open results in a new window

Results 1 - 20 of 200   Result page: **1** 2 3 4 5 6 7 8 9 10   next
Best 200 shown   Relevance scale ☐ ▭ ◪ ◼ ◼

1 Using key object opportunism to collect old objects   ◼
◈ Barry Hayes
November 1991 **ACM SIGPLAN Notices , Conference proceedings on Object-oriented programming
           systems, languages, and applications OOPSLA '91**, Volume 26 Issue 11
Publisher: ACM Press
Full text available: 📄 pdf(1.38 MB)   Additional Information: full citation, references, citings, index terms

2 Implementation techniques: Exploring the barrier to entry: incremental generational garbage collection ◼
◈ for Haskell
A. M. Cheadle, A. J. Field, S. Marlow, S. L. Peyton Jones, R. L. While
October 2004 **Proceedings of the 4th international symposium on Memory management**
Publisher: ACM Press
Full text available: 📄 pdf(458.55 KB)   Additional Information: full citation, abstract, references, index terms

We document the desi n and implementation of a "production" incremental garbage collector for GHC 6.2.It
builds on our earlier work (Non-stop Haskell)that exploited GHC's dynamic dispatch mechanism to hijack
object code pointers so that objects in to-space automatically scavenge themselves when the mutator
attempts to "enter" them. This paper details various optimisations based on code specialisation that remove
the dynamic space,and associated time, overheads that accompanied our earlier sch ...

Keywords: incremental garbage collection, non-stop haskell

3 Non-stop Haskell   ◼
◈ A. M. Cheadle, A. J. Field, S. Marlow, S. L. Peyton Jones, R. L. While
September 2000 **ACM SIGPLAN Notices , Proceedings of the fifth ACM SIGPLAN international conference
           on Functional programming ICFP '00**, Volume 35 Issue 9
Publisher: ACM Press
Full text available: 📄 pdf(557.25 KB)   Additional Information: full citation, abstract, references, citings, index terms

We describe an efficient technique for incorporating Baker's incremental garbage collection algorithm into
the Spineless Tagless G-machine on stock hardware. This algorithm eliminates the stop/go execution
associated with bulk copying collection algorithms, allowing the system to place an upper bound on the
pauses due to garbage collection. The technique exploits the fact that objects are always accessed by
jumping to code rather than being explicitly dereferenced. It works by modifying the entr ...

4 4.2BSD and 4.3BSD as examples of the UNIX system   ◼
◈ John S. Quarterman, Abraham Silberschatz, James L. Peterson
December 1985 **ACM Computing Surveys (CSUR)**, Volume 17 Issue 4
Publisher: ACM Press
Full text available: 📄 pdf(4.07 MB)   Additional Information: full citation, abstract, references, citings, index terms, review

This paper presents an in-depth examination of the 4.2 Berkeley Software Distribution, Virtual VAX-11
Version (4.2BSD), which is a version of the UNIX Time-Sharing System. There are notes throughout on
4.3BSD, the forthcoming system from the University of California at Berkeley. We trace the historical
development of the UNIX system from its conception in 1969 until today, and describe the design principles
that have guided this development. We then present the internal data structures and ...

5
Distributed operating systems

Andrew S. Tanenbaum, Robbert Van Renesse
December 1985 **ACM Computing Surveys (CSUR)**, Volume 17 Issue 4

Publisher: ACM Press

Full text available: pdf(5.49 MB)   Additional Information: full citation, abstract, references, citings, index terms, review

Distributed operating systems have many aspects in common with centralized ones, but they also differ in certain ways. This paper is intended as an introduction to distributed operating systems, and especially to current university research about them. After a discussion of what constitutes a distributed operating system and how it is distinguished from a computer network, various key design issues are discussed. Then several examples of current research projects are examined in some detail ...

6   Multiresolution storage and search in sensor networks

Deepak Ganesan, Ben Greenstein, Deborah Estrin, John Heidemann, Ramesh Govindan
August 2005   **ACM Transactions on Storage (TOS)**, Volume 1 Issue 3

Publisher: ACM Press

Full text available: pdf(1.55 MB)   Additional Information: full citation, abstract, references, index terms

Wireless sensor networks enable dense sensing of the environment, offering unprecedented opportunities for observing the physical world. This article addresses two key challenges in wireless sensor networks: in-network storage and distributed search. The need for these techniques arises from the inability to provide persistent, centralized storage and querying in many sensor networks. Centralized storage requires multihop transmission of sensor data to Internet gateways which can quickly drain b ...

Keywords: Wireless sensor networks, data aging, data storage, drill-down query, multiresolution storage, wavelet processing

7   Storage: An evaluation of multi-resolution storage for sensor networks

Deepak Ganesan, Ben Greenstein, Denis Perelyubskiy, Deborah Estrin, John Heidemann
November 2003 **Proceedings of the 1st international conference on Embedded networked sensor systems**

Publisher: ACM Press

Full text available: pdf(299.34 KB)   Additional Information: full citation, abstract, references, citings

Wireless sensor networks enable dense sensing of the environment, offering unprecedented opportunities for observing the physical world. Centralized data collection and analysis adversely impact sensor node lifetime. Previous sensor network research has, therefore, focused on in network aggregation and query processing, but has done so for applications where the features of interest are known a priori. When features are not known a priori, as is the case with many scientific applications in dens ...

8   A third generation Smalltalk-80 implementation

Patrick J. Caudill, Allen Wirfs-Brock
June 1986   **ACM SIGPLAN Notices , Conference proceedings on Object-oriented programming systems, languages and applications OOPLSA '86**, Volume 21 Issue 11

Publisher: ACM Press

Full text available: pdf(858.89 KB)   .   Additional Information: full citation, abstract, references, citings, index terms

A new, high performance Smalltalk-80™ implementation is described which builds directly upon two previous implementation efforts. This implementation supports a large object space while retaining compatibility with previous Smalltalk-80™ images. The implementation utilizes a interpreter which incorporates a generation based garbage collector and which does not have an object table. This paper describes the design decisions which lead to this implementation and reports preliminar ...

9   Fast detection of communication patterns in distributed executions

Thomas Kunz, Michiel F. H. Seuren
November 1997 **Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research**

Publisher: IBM Press

Full text available: pdf(4.21 MB)   Additional Information: full citation, abstract, references, index terms

Understanding distributed applications is a tedious and difficult task. Visualizations based on process-time diagrams are often used to obtain a better understanding of the execution of the application. The visualization tool we use is Poet, an event tracer developed at the University of Waterloo. However, these diagrams are often very complex and do not provide the user with the desired overview of the application. In our experience, such tools display repeated occurrences of non-trivial commun ...

10   On management of free space in compressed memory systems

Peter A. Franaszek, Philip Heidelberger, Michael Wazlowski
May 1999   **ACM SIGMETRICS Performance Evaluation Review , Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems SIGMETRICS '99**, Volume 27 Issue 1

Publisher: ACM Press

11   The interdisciplinary study of coordination

Thomas W. Malone, Kevin Crowston
March 1994   **ACM Computing Surveys (CSUR),** Volume 26 Issue 1

Publisher: ACM Press
Full text available: pdf(584.94 KB)   Additional Information: full citation, abstract, references, citings, index terms, review

This survey characterizes an emerging research area, sometimes called coordination theory, that focuses on the interdisciplinary study of coordination. Research in this area uses and extends ideas about coordination from disciplines such as computer science, organization theory, operations research, economics, linguistics, and psychology. A key insight of the framework presented here is that coordination can be seen as the process of managing dependencies ...

Keywords: computer-supported cooperative work, coordination, coordination science, coordination theory, groupware

12   Beltway: getting around garbage collection gridlock

Stephen M Blackburn, Richard Jones, Kathryn S. McKinley, J Eliot B Moss
May 2002   **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation PLDI '02,** Volume 37 Issue 5

Publisher: ACM Press
Full text available: pdf(184.50 KB)   Additional Information: full citation, abstract, references, citings, index terms

We present the design and implementation of a new garbage collection framework that significantly generalizes existing copying collectors. The *Beltway* framework exploits and separates object age and incrementality. It groups objects in one or more increments on queues called *belts*, collects belts independently, and collects increments on a belt in first-in-first-out order. We show that Beltway configurations, selected by command line options, act and perform the same as semi-space, ...

Keywords: Java, beltway, copying collection, generational collection

13   An experimental study of renewal-older-first garbage collection

Lars T. Hansen, William D. Clinger
September 2002 **ACM SIGPLAN Notices , Proceedings of the seventh ACM SIGPLAN international conference on Functional programming ICFP '02,** Volume 37 Issue 9

Publisher: ACM Press
Full text available: pdf(143.87 KB)   Additional Information: full citation, abstract, references, citings, index terms

Generational collection has improved the efficiency of garbage collection in fast-allocating programs by focusing on collecting young garbage, but has done little to reduce the cost of collecting a heap containing large amounts of older data. A new generational technique, older-first collection, shows promise in its ability to manage older data.This paper reports on an implementation study that compared two older-first collectors to traditional (younger-first) generational collectors. One of the ...

Keywords: generational garbage collection, older-first

14   CONS should not CONS its arguments, or, a lazy alloc is a smart alloc

Henry G. Baker
March 1992   **ACM SIGPLAN Notices,** Volume 27 Issue 3

Publisher: ACM Press
Full text available: pdf(1.52 MB)   Additional Information: full citation, abstract, index terms

Lazy allocation is a model for allocating objects on the execution stack of a high-level language which does not create dangling references. Our model provides safe transportation into the heap for objects that may survive the deallocation of the surrounding stack frame. Space for objects that do not survive the deallocation of the surrounding stack frame is reclaimed without additional effort when the stack is popped. Lazy allocation thus performs a first-level garbage collection, and if ...

15   Generational stack collection and profile-driven pretenuring

Perry Cheng, Robert Harper, Peter Lee
May 1998   **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation PLDI '98,** Volume 33 Issue 5

Publisher: ACM Press
Full text available: pdf(1.56 MB)   Additional Information: full citation, abstract, references, citings, index terms

This paper presents two techniques for improving garbage collection performance: generational stack collection and profile-driven pretenuring. The first is applicable to stack-based implementations of functional languages while the second is useful for any generational collector. We have implemented both techniques in a generational collector used by the TIL compiler (Tarditi, Morrisett, Cheng, Stone, Harper, and Lee 1996), and have observed decreases in garbage collection times of as much as 70 ...

16 Dynamic adaptive pre-tenuring

Timothy L. Harris

October 2000   **ACM SIGPLAN Notices , Proceedings of the 2nd international symposium on Memory management ISMM '00,** Volume 36 Issue 1

Publisher: ACM Press

Full text available: pdf(1.16 MB)   Additional Information: full citation, abstract, citings, index terms

In a generational garbage collector, a *pre-tenured* object is one that is allocated directly in the old generation. Pre-tenuring long-lived objects reduces the number of times that they are scanned or copied during garbage collection. Previous work has investigated pre-tenuring based on off-line analysis of execution traces. This paper builds on that work by presenting a *dynamic* technique in which the decision to pre-tenure a particular kind of object is taken at run-time. This a ...

17 Efficient memory management in a merged heap/stack prolog machine

Xining Li

September 2000   **Proceedings of the 2nd ACM SIGPLAN international conference on Principles and practice of declarative programming**

Publisher: ACM Press

Full text available: pdf(553.36 KB)   Additional Information: full citation, references, index terms

18 Mark-copy: fast copying GC with less space overhead

Narendran Sachindran, J. Eliot, B. Moss

October 2003   **ACM SIGPLAN Notices , Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications OOPSLA '03,** Volume 38 Issue 11

Publisher: ACM Press

Full text available: pdf(297.93 KB)   Additional Information: full citation, abstract, references, citings, index terms

Copying garbage collectors have a number of advantages over non-copying collectors, including cheap allocation and avoiding fragmentation. However, in order to provide completeness (the guarantee to reclaim each garbage object eventually), standard copying collectors require space equal to twice the size of the maximum live data for a program. We present a *mark-copy* collection algorithm (MC) that extends generational copying collection and significantly reduces the heap space required to ...

Keywords: Java, copying collector, generational collector, mark-copy, mark-sweep

19 A generational mostly-concurrent garbage collector

Tony Printezis, David Detlefs

October 2000   **ACM SIGPLAN Notices , Proceedings of the 2nd international symposium on Memory management ISMM '00,** Volume 36 Issue 1

Publisher: ACM Press

Full text available: pdf(1.67 MB)   Additional Information: full citation, abstract, citings, index terms

This paper reports our experiences with a mostly-concurrent incremental garbage collector, implemented in the context of a high performance virtual machine for the Java™ programming language. The garbage collector is based on the "mostly parallel" collection algorithm of Boehm *et al.* and can be used as the old generation of a generational memory system. It overloads efficient write-barrier code already generated to support generational garbage collection to also ident ...

20 The GemStone object database management system

Paul Butterworth, Allen Otis, Jacob Stein

October 1991   **Communications of the ACM,** Volume 34 Issue 10

Publisher: ACM Press

Full text available: pdf(6.60 MB)   Additional Information: full citation, references, citings, index terms

Keywords: GemStone, database management systems, object-oriented

Feedback Report a problem Satisfaction survey

Terms used frame and object and allocate or allocation and older frame   Found **40,890** of **178,880**

Sort results by relevance ▾

Display results expanded form ▾

🔖 Save results to a Binder

❓ Search Tips

☐ Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 200   Result page: **1** 2 3 4 5 6 7 8 9 10   next
Best 200 shown   Relevance scale ☐ ◻ ◼ ◼ ◼

1 Older-first garbage collection in practice: evaluation in a Java Virtual Machine   ◼
Darko Stefanović, Matthew Hertz, Stephen M. Blackburn, Kathryn S. McKinley, J. Eliot B. Moss
June 2002   **ACM SIGPLAN Notices , Proceedings of the 2002 workshop on Memory system
              performance MSP '02**, Volume 38 Issue 2 supplement
Publisher: ACM Press
Full text available: 📄 pdf(1.15 MB)   Additional Information: full citation, abstract, references, citings

Until recently, the best performing copying garbage collectors used a generational policy which repeatedly
collects the very youngest objects, copies any survivors to an older space, and then infrequently collects the
older space. A previous study that used garbage-collection simulation pointed to potential improvements by
using an *Older-First* copying garbage collection algorithm. The Older-First algorithm sweeps a fixed-sized
window through the heap from older to younger objects, and avo ...

2 CONS should not CONS its arguments, or, a lazy alloc is a smart alloc   ◼
Henry G. Baker
March 1992   **ACM SIGPLAN Notices**, Volume 27 Issue 3
Publisher: ACM Press
Full text available: 📄 pdf(1.52 MB)   Additional Information: full citation, abstract, index terms

Lazy allocation is a model for allocating objects on the execution stack of a high-level language which does
not create dangling references. Our model provides safe transportation into the heap for objects that may
survive the deallocation of the surrounding stack frame. Space for objects that do not survive the
deallocation of the surrounding stack frame is reclaimed without additional effort when the stack is popped.
Lazy allocation thus performs a first-level garbage collection, and if ...

3 Beltway: getting around garbage collection gridlock   ◼
Stephen M Blackburn, Richard Jones, Kathryn S. McKinley, J Eliot B Moss
May 2002   **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on
             Programming language design and implementation PLDI '02**, Volume 37 Issue 5
Publisher: ACM Press
Full text available: 📄 pdf(184.50 KB)   Additional Information: full citation, abstract, references, citings, index terms

We present the design and implementation of a new garbage collection framework that significantly
generalizes existing copying collectors. The *Beltway* framework exploits and separates object age and
incrementality. It groups objects in one or more increments on queues called *belts*, collects belts
independently, and collects increments on a belt in first-in-first-out order. We show that Beltway
configurations, selected by command line options, act and perform the same as semi-space, ...

Keywords: Java, beltway, copying collection, generational collection

4 Generational stack collection and profile-driven pretenuring   ◼
Perry Cheng, Robert Harper, Peter Lee
May 1998   **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on
             Programming language design and implementation PLDI '98**, Volume 33 Issue 5
Publisher: ACM Press
Full text available: 📄 pdf(1.56 MB)   Additional Information: full citation, abstract, references, citings, index terms

This paper presents two techniques for improving garbage collection performance: generational stack
collection and profile-driven pretenuring. The first is applicable to stack-based implementations of functional
languages while the second is useful for any generational collector. We have implemented both techniques in
a generational collector used by the TIL compiler (Tarditi, Morrisett, Cheng, Stone, Harper, and Lee 1996),

and have observed decreases in garbage collection times of as much as 70 ...

5 Systems: Multi-fidelity storage

Padmanabhan Pillai, Yan Ke, Jason Campbell

October 2004 **Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks**

Publisher: ACM Press

Full text available: pdf(184.80 KB) Additional Information: full citation, abstract, references, index terms

Imaging sensors are inherently high bandwidth devices, and applications which store image data often encounter disk or memory limits. Commonly, upon reaching such a limit, storage systems will cease sampling or overwrite existing data in an oldest-first fashion. For most applications, neither approach is optimal. We introduce a flexible, policy-based, on-line algorithm for maximizing the application-specific value of data retained on a read/write/erase storage medium such as a hard disk or fl ...

Keywords: variable fidelity, video surveillance, weighted fair-share storage

6 Efficient memory management in a merged heap/stack prolog machine

Xining Li

September 2000 **Proceedings of the 2nd ACM SIGPLAN international conference on Principles and practice of declarative programming**

Publisher: ACM Press

Full text available: pdf(553.36 KB) Additional Information: full citation, references, index terms

7 Non-stop Haskell

A. M. Cheadle, A. J. Field, S. Marlow, S. L. Peyton Jones, R. L. While

September 2000 **ACM SIGPLAN Notices , Proceedings of the fifth ACM SIGPLAN international conference on Functional programming ICFP '00**, Volume 35 Issue 9

Publisher: ACM Press

Full text available: pdf(557.25 KB) Additional Information: full citation, abstract, references, citings, index terms

We describe an efficient technique for incorporating Baker's incremental garbage collection algorithm into the Spineless Tagless G-machine on stock hardware. This algorithm eliminates the stop/go execution associated with bulk copying collection algorithms, allowing the system to place an upper bound on the pauses due to garbage collection. The technique exploits the fact that objects are always accessed by jumping to code rather than being explicitly dereferenced. It works by modifying the entr ...

8 Implementation techniques: Exploring the barrier to entry: incremental generational garbage collection for Haskell

A. M. Cheadle, A. J. Field, S. Marlow, S. L. Peyton Jones, R. L. While

October 2004 **Proceedings of the 4th international symposium on Memory management**

Publisher: ACM Press

Full text available: pdf(458.55 KB) Additional Information: full citation, abstract, references, index terms

We document the desi n and implementation of a "production" incremental garbage collector for GHC 6.2.It builds on our earlier work (Non-stop Haskell)that exploited GHC's dynamic dispatch mechanism to hijack object code pointers so that objects in to-space automatically scavenge themselves when the mutator attempts to "enter" them. This paper details various optimisations based on code specialisation that remove the dynamic space,and associated time, overheads that accompanied our earlier sch ...

Keywords: incremental garbage collection, non-stop haskell

9 Automated discovery of scoped memory regions for real-time Java

Morgan Deters, Ron K. Cytron

June 2002 **ACM SIGPLAN Notices , Proceedings of the 3rd international symposium on Memory management ISMM '02**, Volume 38 Issue 2 supplement

Publisher: ACM Press

Full text available: pdf(227.49 KB) Additional Information: full citation, abstract, references, citings, index terms

Advances in operating systems and languages have brought the ideal of reasonably-bounded execution time closer to developers who need such assurances for real-time and embedded systems applications. Recently, extensions to the Java libraries and virtual machine have been proposed in an emerging standard, which provides for specification of release times, execution costs, and deadlines for a restricted class of threads. To use such features, the code executing in the thread must never reference s ...

Keywords: garbage collection, memory management, real-time Java, regions, trace-based analysis

10 Dynamic adaptive pre-tenuring

Timothy L. Harris

October 2000 **ACM SIGPLAN Notices , Proceedings of the 2nd international symposium on Memory management ISMM '00,** Volume 36 Issue 1

Publisher: ACM Press

Full text available: pdf(1.16 MB)   Additional Information: full citation, abstract, citings, index terms

> In a generational garbage collector, a *pre-tenured* object is one that is allocated directly in the old generation. Pre-tenuring long-lived objects reduces the number of times that they are scanned or copied during garbage collection. Previous work has investigated pre-tenuring based on off-line analysis of execution traces. This paper builds on that work by presenting a *dynamic* technique in which the decision to pre-tenure a particular kind of object is taken at run-time. This a ...

11 Contaminated garbage collection

Dante J. Cannarozzi, Michael P. Plezbert, Ron K. Cytron

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation PLDI '00,** Volume 35 Issue 5

Publisher: ACM Press

Full text available: pdf(559.20 KB)   Additional Information: full citation, abstract, references, citings, index terms

> We describe a new method for determining when an object can be garbage collected. The method does not require marking live objects. Instead, each object X is dynamically associated with a stack frame M, such that Xis collectable when M pops. Because X could have been dead earlier, our method is conservative. Our results demonstrate that the method nonetheless identifies a large percentag ...

12 Parallel execution of prolog programs: a survey

Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson, Manuel V. Hermenegildo

July 2001 **ACM Transactions on Programming Languages and Systems (TOPLAS),** Volume 23 Issue 4

Publisher: ACM Press

Full text available: pdf(1.95 MB)   Additional Information: full citation, abstract, references, citings, index terms

> Since the early days of logic programming, researchers in the field realized the potential for exploitation of parallelism present in the execution of logic programs. Their high-level nature, the presence of nondeterminism, and their referential transparency, among other characteristics, make logic programs interesting candidates for obtaining speedups through parallel execution. At the same time, the fact that the typical applications of logic programming frequently involve irregular computatio ...

> **Keywords:** Automatic parallelization, constraint programming, logic programming, parallelism, prolog

13 Generating object lifetime traces with Merlin

Matthew Hertz, Stephen M. Blackburn, J. Eliot B. Moss, Kathryn S. McKinley, Darko Stefanović

May 2006 **ACM Transactions on Programming Languages and Systems (TOPLAS),** Volume 28 Issue 3

Publisher: ACM Press

Full text available: pdf(1.31 MB)   Additional Information: full citation, abstract, references, index terms

> Programmers are writing a rapidly growing number of programs in object-oriented languages, such as Java and C#, that require garbage collection. Garbage collection traces and simulation speed up research by enabling deeper understandings of object lifetime behavior and quick exploration and design of new garbage collection algorithms. When generating perfect traces, the *brute-force* method of computing object lifetimes requires a whole-heap garbage collection at every potential collect ...

> **Keywords:** Garbage collection, object lifetime analysis, trace design, trace generation

14 The runtime environment for Scheme, a Scheme implementation on the 88000

Steven R. Vegdahl, Uwe F. Pleban

April 1989 **ACM SIGARCH Computer Architecture News , Proceedings of the third international conference on Architectural support for programming languages and operating systems ASPLOS-III,** Volume 17 Issue 2

Publisher: ACM Press

Full text available: pdf(1.22 MB)   Additional Information: full citation, abstract, references, citings, index terms, review

> We are implementing a Scheme development system for the Motorola 88000. The core of the implementation is an optimizing native code compiler, together with a carefully designed runtime system. This paper describes our experiences with the 88000 as a target architecture. We focus on the design decisions concerning the runtime system, particularly with respect to data type representations, tag checking, procedure calling protocol, generic arithmetic, and the handling of continuations. We also ...

15

An experimental study of renewal-older-first garbage collection
Lars T. Hansen, William D. Clinger
September 2002 **ACM SIGPLAN Notices , Proceedings of the seventh ACM SIGPLAN international conference on Functional programming ICFP '02,** Volume 37 Issue 9
Publisher: ACM Press
Full text available: pdf(143.87 KB)   Additional Information: full citation, abstract, references, citings, index terms

Generational collection has improved the efficiency of garbage collection in fast-allocating programs by focusing on collecting young garbage, but has done little to reduce the cost of collecting a heap containing large amounts of older data. A new generational technique, older-first collection, shows promise in its ability to manage older data.This paper reports on an implementation study that compared two older-first collectors to traditional (younger-first) generational collectors. One of the ...

Keywords: generational garbage collection, older-first

16 A comparative performance evaluation of write barrier implementation
Antony L. Hosking, J. Eliot B. Moss, Darko Stefanovic
October 1992 **ACM SIGPLAN Notices , conference proceedings on Object-oriented programming systems, languages, and applications OOPSLA '92,** Volume 27 Issue 10
Publisher: ACM Press
Full text available: pdf(2.48 MB)   Additional Information: full citation, references, citings, index terms

17 Speedy wireless: Horde: separating network striping policy from mechanism
Asfandyar Qureshi, John Guttag
June 2005 **Proceedings of the 3rd international conference on Mobile systems, applications, and services MobiSys '05**
Publisher: ACM Press
Full text available: pdf(284.87 KB)   Additional Information: full citation, abstract, references

Inverse multiplexing, or network striping, allows the construction of a high-bandwidth virtual channel from a collection of multiple low-bandwidth network channels. Striping systems usually employ an immutable packet scheduling policy and allow applications to be oblivious of the way in which packets are routed to specific network channels. Though this is appropriate for many applications, other applications can benefit from an approach that explicitly involves the application in the dynamic det ...

18 Sapphire: copying GC without stopping the world
Richard L. Hudson, J. Eliot B. Moss
June 2001 **Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande**
Publisher: ACM Press
Full text available: pdf(899.45 KB)   Additional Information: full citation, abstract, references, citings, index terms

Many concurrent garbage collection (GC) algorithms have been devised, but few have been implemented and evaluated, particularly for the Java programming language. Sapphire is an algorithm we have devised for concurrent copying GC. Sapphire stresses minimizing the amount of time any given application thread may need to block to support the collector. In particular, Sapphire is intended to work well in the presence of a large number of application threads, on small- to medium-scale shared memor ...

19 Mark-copy: fast copying GC with less space overhead
Narendran Sachindran, J. Eliot, B. Moss
October 2003 **ACM SIGPLAN Notices , Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications OOPSLA '03,** Volume 38 Issue 11
Publisher: ACM Press
Full text available: pdf(297.93 KB)   Additional Information: full citation, abstract, references, citings, index terms

Copying garbage collectors have a number of advantages over non-copying collectors, including cheap allocation and avoiding fragmentation. However, in order to provide completeness (the guarantee to reclaim each garbage object eventually), standard copying collectors require space equal to twice the size of the maximum live data for a program. We present a *mark-copy* collection algorithm (MC) that extends generational copying collection and significantly reduces the heap space required to ...

Keywords: Java, copying collector, generational collector, mark-copy, mark-sweep

20 Real-time shading
Marc Olano, Kurt Akeley, John C. Hart, Wolfgang Heidrich, Michael McCool, Jason L. Mitchell, Randi Rost
August 2004 **Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04**
Publisher: ACM Press

Real-time procedural shading was once seen as a distant dream. When the first version of this course was offered four years ago, real-time shading was possible, but only with one-of-a-kind hardware or by combining the effects of tens to hundreds of rendering passes. Today, almost every new computer comes with graphics hardware capable of interactively executing shaders of thousands to tens of thousands of instructions. This course has been redesigned to address today's real-time shading capabili ...